Transferring a Petabyte in a Day

Rajkumar Kettimuthu[a], Zhengchun Liu[a,*], David Wheeler[d], Ian Foster[a,b], Katrin Heitmann[a,c], Franck Cappello[a]

[a] *Mathematics & Computer Science, Argonne National Laboratory, Lemont, IL 60439, USA*
[b] *Department of Computer Science, University of Chicago, Chicago, IL 60637, USA*
[c] *High Energy Physics, Argonne National Laboratory, Lemont, IL 60439, USA*
[d] *National Center for Supercomputing Applications, University Of Illinois at Urbana-Champaign*

Abstract

Extreme-scale simulations and experiments can generate large amounts of data, whose volume can exceed the compute and/or storage capacity at the simulation or experimental facility. With the emergence of ultra-high-speed networks, it becomes feasible to consider pipelined approaches in which data are passed to a remote facility for analysis. Here we examine the case of an extreme-scale cosmology simulation that, when run on a large fraction of a leadership-scale computer, generates data at a rate of one petabyte per elapsed day. Writing those data to disk is inefficient and impractical, and in situ analysis poses its own difficulties. Thus we implement a pipeline in which data are generated on one supercomputer and then transferred, as they are generated, to a remote supercomputer for analysis. We use the Swift scripting language to instantiate this pipeline across Argonne National Laboratory and the National Center for Supercomputing Applications, which are connected by a 100 Gb/s network, and demonstrate that by using the Globus transfer service we can achieve a sustained rate of 93 Gb/s over a 24-hour period and thus achieve our performance goal of one petabyte moved in 24 hours. This paper describes the methods used and summarizes the lessons learned in this demonstration.

Keywords: Wide area data transfer, GridFTP, Large data transfer, Cosmology workflow, Pipeline

1. Introduction

Extreme-scale scientific simulations and experiments can generate much more data than can be stored and analyzed efficiently at a single site. For example, a single trillion-particle simulation with the Hardware/Hybrid Accelerated Cosmology Code (HACC) cosmology code [1] generates 20 PB of raw data (500 snapshots, each 40 TB), which is more than petascale systems such as Mira and Blue Waters can store in their file systems. Meanwhile, much as scientific instruments are optimized for specific objectives, both computational infrastructure and codes become more specialized as we reach the end of Moore's law. For example, one version of the HACC simulation code is optimized for Argonne National Laboratory's Mira supercomputer, on which it can scale to millions of cores, while the National Center for Supercomputer Applications (NCSA)'s Blue Waters computer is an excellent system for data analysis, due to its large memory (1.5 PB) and 4000+ GPU accelerators.

To both overcome storage limitations and enable the coordinated use of these two specialized systems, we demonstrated pipelined remote analysis of HACC simulation data, as shown in Figure 1. A state-of-the-art, 29-billion-particle cosmology simulation combining high spatial and temporal resolution in a large cosmological volume was performed on Mira. As this simulation ran, each of 500 temporal snapshots was transmitted as it was produced to NCSA by using the Globus transfer service [2]. In total, this workflow

* Corresponding author at: Bldg. 240, Argonne National Laboratory, 9700 S. Cass Avenue, Lemont, IL 60439.
Email addresses: kettimut@anl.gov (Rajkumar Kettimuthu), zhengchun.liu@anl.gov (Zhengchun Liu), foster@anl.gov (Ian Foster), heitmann@anl.gov (Katrin Heitmann), cappello@mcs.anl.gov (Franck Cappello)

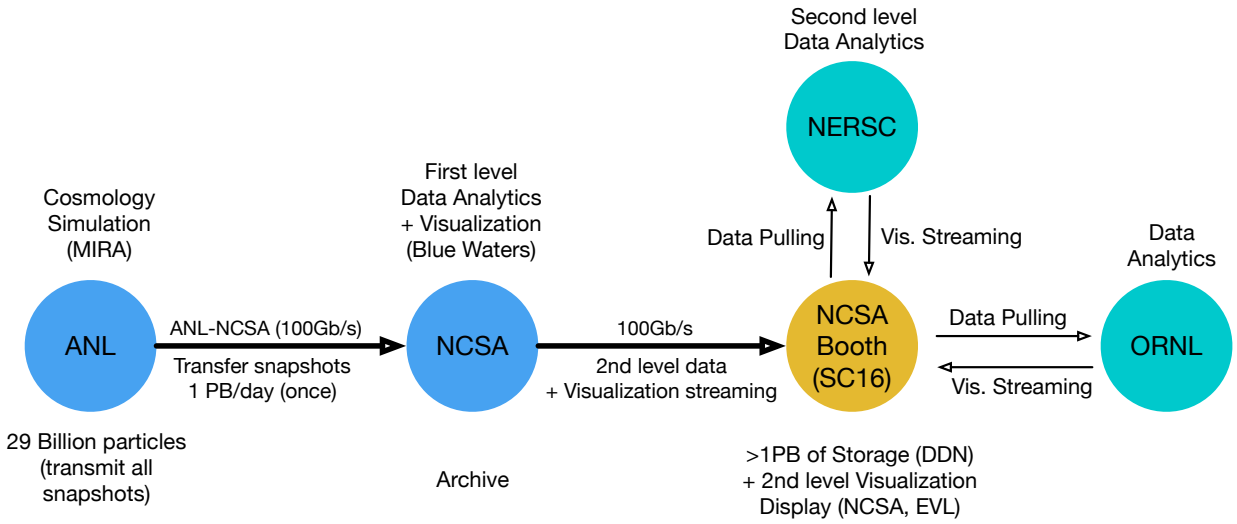

Figure 1: Pipelined execution of a cosmology workflow that involves a mix of streaming scenarios, ranging from sustained ∼100 Gb/s over a 24 hour period to high-bandwidth bursts during interactive analysis sessions.

moved 1 PB in 24 hours from ALCF to NCSA, requiring an average end-to-end rate of ∼93 Gb/s. In this paper, we describe how we achieved this feat, including the experiments performed to gather insights on tuning parameters, data organization, and the lessons learned.

A first level of data analysis and visualization was performed as snapshots arrived at NCSA using the GPU partition of Blue Waters. After the snapshot arrived on Blue Waters, an analysis task submission was triggered. We note that the analysis tasks have to be carried out sequentially (time step after time step): information from the previous time snapshot is captured for the analysis of the next time snapshot in order to enable detailed tracking of the evolution of structures. The workflow system therefore had to be carefully designed to resubmit any unsuccessful analysis job and had to wait for an analysis job to finish before starting the next one. The output data (half the size of the input data) was then sent to the SC16 NCSA booth to allow access to and sharing of the resulting data from remote sites. The whole experiment was orchestrated by the Swift parallel scripting language [3]. In previous simulations, scientists were able to analyze only ∼100 snapshots because of infrastructure limitations.

This experiment achieved two objectives never accomplished before: (1) running a state-of-the-art cosmology simulation and analyzing all snapshots (currently only one in every five or 10 snapshots is stored or communicated); and (2) combining two different types of systems (simulation on Mira and data analytics on Blue Waters) that are geographically distributed and belong to different administrative domains to run an extreme-scale simulation and analyze the output in a pipelined fashion.

The work presented here is also unique in two other respects. First, while many previous studies have varied transfer parameters such as concurrency and parallelism to improve data transfer performance [4, 5, 6], we also demonstrate the value of varying the file size used for data transfer, which provides additional flexibility for optimization. Second, we demonstrate these methods in the context of dedicated data transfer nodes and a 100 Gb/s network circuit.

The rest of the paper is as follows. We first introduce, in §2, the science case and the environment in which we perform these transfers. In §3 we describe the tests to find the optimal transfer parameters. Then we summarize the performance of the transfers during the pipelined simulation and analysis experiments, and our experiences with checksum-enabled transfers in §4. Retrospecting on the SC16 demo, we propose in §5 an analytical model to identify the optimal file size and show that it can help improve the performance of checksum-enabled transfers significantly. Finally, we review related work in §6, and in §7 summarize our work.

2. Science case and demonstration environment

We first describe the science case and the environment used for the demonstration.

2.1. Science case

The nature of cosmology precludes carrying out controlled experiments in the laboratory. So, cosmologists must observe the Universe and interpret the results to understand the history and content of the Universe. Large observational surveys are carried out using telescopes. These surveys are becoming complex as telescopes reach deeper into space, mapping out the distributions of galaxies at farther distances. Cosmological simulations that track the detailed evolution of structure in the Universe over time are essential for interpreting these surveys. Cosmologists vary the fundamental physics in the simulations, evaluate resulting measurements in a controlled way, and then predict new phenomena. They also model systematic errors in the data, mimic inaccuracies due to limiting telescope and sensor artifacts, and understand how these limitations can influence our scientific results. In order to achieve the high-quality simulations, high temporal and spatial resolution are of utmost importance. They need to track the evolution of small overdensities in detail and follow how they evolve into larger structures. Events early in the life of such a structure will determine what kind of galaxy it will host later, controlling, for example, the brightness, color, and morphology of the galaxy. Current (and next-generation) supercomputers (will) allow them to attain high spatial resolution in large cosmological volumes by simulating trillions of tracer particles. But the supercomputer on which the simulation is carried out might not be—and usually is not—the optimal system for data analysis.

2.2. Demonstration environment

For the demonstration, we ran a large simulation at the Argonne Leadership Computing Facility (ALCF), moved the raw simulation output to NCSA, and ran an analysis program on the Blue Waters supercomputer. The simulation evolved 3072^3 (=29 billion) particles in a simulation box of volume $(512h^{-1}Mpc)^3$. This led to an approximate mass resolution (mass of individual particles) of mp=$3.8*10^8$ h^{-1}Msun.

Each snapshot holds 1.2 TB. The source of the data was the GPFS parallel file system on the Mira supercomputer at Argonne, and the destination was the Lustre parallel file system on the Blue Waters supercomputer at NCSA. Argonne and NCSA have 12 and 28 data transfer nodes (DTNs) [7], respectively, dedicated for wide-area data transfer. Each DTN runs a GridFTP server. We chose to use Globus to orchestrate our data transfers in order to get automatic fault recovery and load balancing among the available GridFTP servers on both ends.

3. Exploration of tunable parameters

Concurrency, parallelism, and pipelining are three key performance optimization mechanisms for Globus GridFTP transfers. Concurrency uses multiple GridFTP server processes at the source and destination, where each process transfers a separate file, and thus provides for concurrency at the file system I/O, CPU core, and network levels. Parallelism is a network-level optimization that uses multiple socket connections to transfer chunks of a file in parallel from a single-source GridFTP server process to a single-destination GridFTP server process. Pipelining speeds up lots of tiny files by sending multiple FTP commands to a single GridFTP server process without waiting for the first command's response. This reduces latency between file transfers in a single GridFTP server process. Concurrency and parallelism are illustrated in Figure 2 and pipelining is illustrated in Figure 3.

In order to find the best application tunable parameters, we fixed the average file size to be ∼4 GB and evaluated different combinations of three Globus GridFTP parameters: concurrency, parallelism, and pipeline depth.

Figure 4 shows the achieved throughput as a function of parallelism for different concurrencies and pipeline depths. It is clear that parallelism does not provide any obvious improvement in performance. Our conjecture is that this is because the round-trip time between source DTNs and destination DTNs is pretty small (around 6ms). Figure 5 shows the achieved throughput as a function of concurrency for different

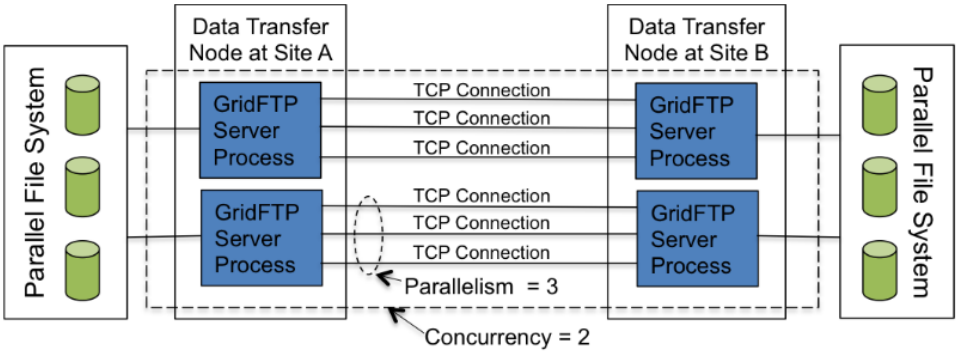

Figure 2: Illustration of concurrency and parallelism in Globus GridFTP.

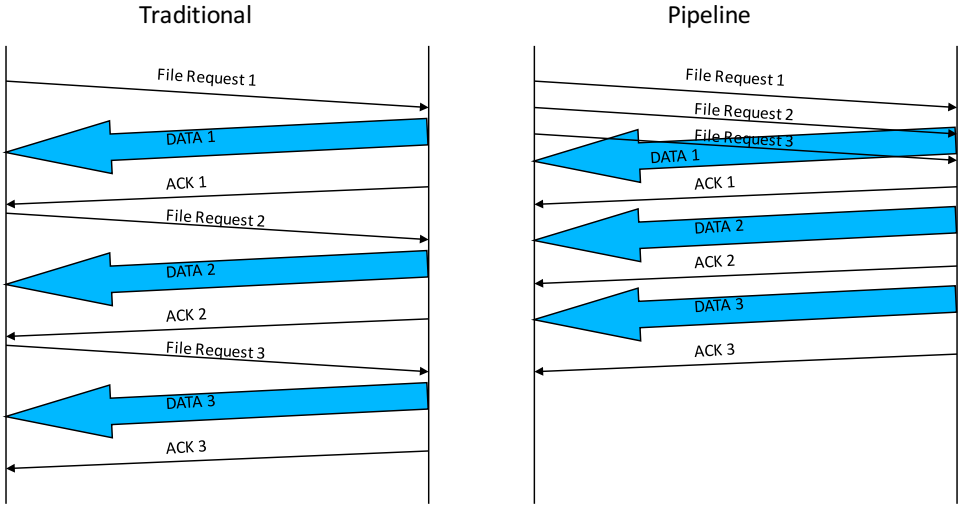

Figure 3: Illustration of pipelining in Globus GridFTP.

pipeline depth values. We omitted parallelism in this comparison as it does not have much of an impact (Figure 4).

From Figure 5, we see that the throughput increases with increasing concurrency, especially for small pipeline depths. Each DTN at ALCF has only a 10 Gb/s network interface card and a transfer needs to use at least 10 DTNs to achieve the desired rate of 1 PB in a day or sustained ∼93 Gb/s. In order for a transfer to use 10 DTNs, the concurrency has to be at least 10. In order for each DTN to drive close to 10 Gb/s (read data from the storage at 10 Gb/s and send data on the network at the same rate), many (or all) DTN cores need to be used. In this case, each DTN has 8 cores and thus a concurrency of at least 96 is needed to use all cores. This explains why a concurrency of 128 gives the highest performance.

We also see in Figure 5 that increasing pipeline depth reduces performance. This is because of the Globus policy that was designed for regular data transfers (i.e., transfer size is not as big as this case, the endpoints and network is not as powerful as it is in this case). Specifically, Globus splits multi-file transfers into batches of 1,000 files and treats each batch like an independent transfer request. This policy was put in place to control the total number of concurrent file transfers and thus the memory load on servers. For example, if we use pipeline depth of 8, the maximum concurrency can only be $\lfloor \frac{1000}{16} \rfloor = 62$, which is why concurrency with 64 and 128 got the same performance in Figure 5.

Similarly, when pipeline depth is 16, the actual concurrency will be $\lfloor \frac{1000}{32} \rfloor = 31$, and thus transfers with

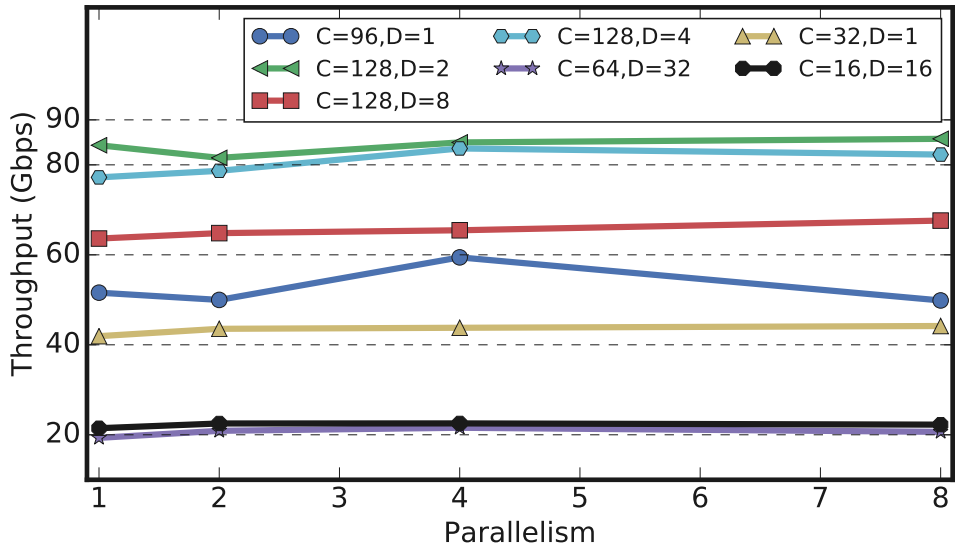

Figure 4: Data transfer performance versus parallelism with different concurrency (C) and pipeline depth(D).

concurrency greater that 32 get the same performance as those with 32 in Figure 5. Therefore, the optimal pipeline depth for our use case is one, because pipelining is good for transferring tiny files (when the elapsed transfer time for one file is less that the RTT between the client and the source endpoint) but not for larger files.

4. Experiences transferring the data

As mentioned before, we split each 1.2 TB snapshot into 256 files of approximately equal size. We determined that transferring 64 or 128 files concurrently, with a total of 128 or 256 TCP streams, yielded the maximum transfer rate. We achieved an average disk-to-disk transfer rate of 92.4 Gb/s (or 1 PB in 24 hours and 3 minutes): 99.8% of our goal of 1 PB in 24 hours, when the end-to-end verification of data integrity in Globus is disnabled. When the end-to-end verification of data integrity in Globus is enabled, we achieved an average transfer rate of only 72 Gb/s (or 1 PB in 30 hours and 52 minutes).

The Globus approach to checksum verification is motivated by the observations that the 16-bit TCP checksum is inadequate for detecting data corruption during communication [8, 9], and that corruption can furthermore occur during file system operations [10]. Globus pipelines the transfer and checksum computation; that is, the checksum computation of the ith file happens in parallel with the transfer of the $(i+1)$th file. Data are read twice at the source storage system (once for transfer and once for checksum) and written once (for transfer) and read once (for checksum) at the destination storage system. In order to achieve the desired rate of 93 Gb/s for checksum-enabled transfers, then in the absence of checksum failures, 186 Gb/s of read bandwidth from the source storage system and 93 Gb/s write bandwidth and 93 Gb/s of read bandwidth concurrently from the destination storage system are required. If checksum verification failures occur (i.e., one or more files are corrupted during the transfer), even more storage I/O bandwidth, CPU resources, and network bandwidth are required to achieve the desired rate. Figure 6 shows the overall transfer throughout, as determined via SNMP network monitoring, and the DTN CPU utilization, when performing transfers using the optimal parameters that we identified.

We see that transfers without integrity checking (marked by dashed line boxes in Figure 6) can sustain rates close to our environment's theoretical bandwidth of 100 Gb/s, with little CPU utilization. However, if integrity checking is enabled (solid line boxes in Figure 6), CPU utilization increases significantly and it is hard to get close to the theoretical bandwidth continuously. We note that: (1) the network is not dedicated to this experiment and so some network bandwidth was unavoidably consumed by other programs; (2) we

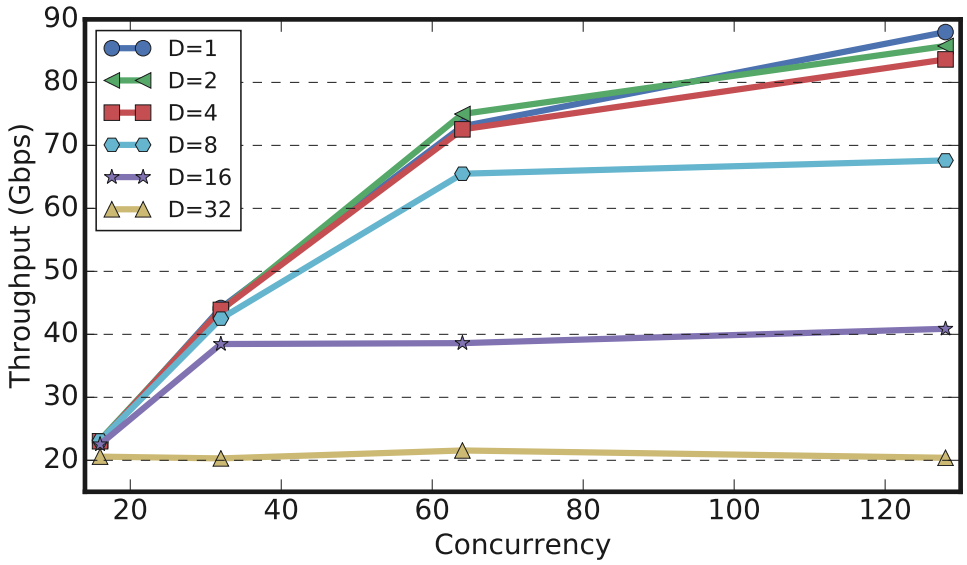

Figure 5: Data transfer performance versus concurrency for different pipeline depth values.

used the same optimal tunable parameters (concurrency and parallelism) and the same file size for transfers with and without checksum in order to make sure that the only difference between the two cases is data verification. (3) there are transfers with non-optimal parameters, performed as part of our explorations, running during other times (outside the boxes) in Figure 6.

4.1. Checksum failures

Globus restarts failed or interrupted transfers from the last checkpoint in order to avoid retransmission costs. However, in the case of a checksum error, it retransmits the entire erroneous file. About 5% of our transfers experienced checksum failure. It can either be caused by network corruption, source storage error or destination storage error. Since the data integrity is verified after the whole file has been transferred to destination, a retransmission must be done if their checksum does not match. For a given transfer, the number of failure represents number of retransferred files. Obviously the transfer throughput will go down if there are too many failures. We show the transfer throughput versus failures in Figure 7.

Assume that a transfer contains N files, each of x bytes, and there are n checksum verification failures. Thus, n files are retransferred and the total bytes transferred will then be $(N+n)x$. If we assume that the end-to-end throughout is R_{e2e}, the actual transfer time T_{trs} will be

$$T_{\text{trs}} = \frac{x\,(N+n)}{R_{\text{e2e}}} \tag{1}$$

Thus, the effective throughout R_{trs} to the transfer users , i.e., it takes T_{trs} seconds to transfer Nx bytes, will be

$$R_{\text{trs}} = \frac{Nx}{T_{\text{trs}}} = \frac{NR_{\text{e2e}}}{N+n} \tag{2}$$

We note that transfers in Figure 7 have different concurrency, parallelism, pipeline depth, and average file size, and thus their R_{e2e} are different. If we only look at the transfers with similar concurrency, the shape in Figure 7 fits well with Equation 2.

5. Retrospective analysis: A model-based approach to finding the optimal number of files

The ability to control the file size (and thus number of files) in the dataset is a key flexibility in this use case. Thus, while we used a file size of 4 GB in our experiments, based on limited exploration and intuition, we

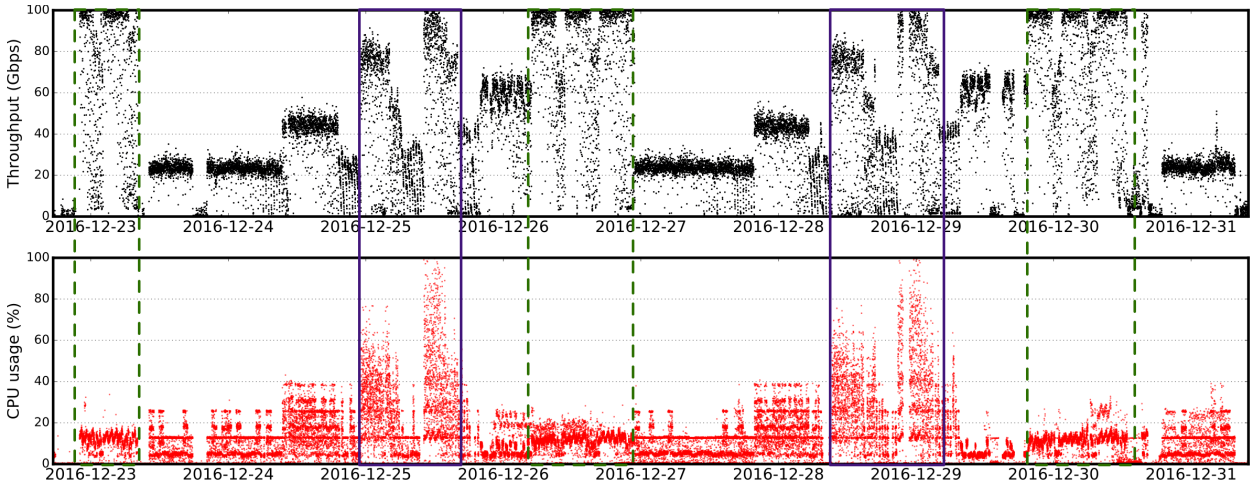

Figure 6: Data transfer throughput vs. DTN CPU usage over an eight-day period. We highlight two periods in which our data transfers were occurring with checksum computations (periods delineated by solid line box) and three periods in which transfers were occurring without checksum computations (dashed line boxes).

realized in retrospect that we could have created a model to identify the optimal file size. Here, we present follow up work in which we develop and apply such a model.

In developing a model of file transfer plus checksum costs, we start with a simple linear model of transfer time for a single file:

$$T_{\text{trs}} = a_{\text{trs}}x + b_{\text{trs}}, \tag{3}$$

where a_{trs} is the unit transfer time, b_{trs} the transfer startup cost, and x the file size. Similarly, we model the time to verify file integrity as:

$$T_{\text{ck}} = a_{\text{ck}}x + b_{\text{ck}}, \tag{4}$$

where a_{ck} is the unit checksum time, b_{ck} the checksum startup cost, and x the file size.

The time to transfer *and* checksum a file is not the simple sum of these two equations because, as shown in Figure 8, Globus GridFTP pipelines data transfers and associated checksum computations. Note how the data transfer and file integrity verification computations overlap. Thus, assuming no file system contention and that unit checksum time is less than unit transfer time (we verified that this is indeed the case in our environment), the total time T to transfer n files with one GridFTP process, each of size x, is the time required for N file transfers and one checksum, i.e.,

$$T = nT_{\text{trs}} + T_{\text{ck}} + b_{\text{trs}} = n(xa_{\text{trs}} + b_{\text{trs}}) + xa_{\text{ck}} + b_{\text{ck}} + b_{\text{trs}}, \tag{5}$$

where b_{trs} is the transfer startup cost (e.g., time to establish the Globus control channel). Let us now assume that concurrency = cc, S denotes the bytes to be transferred in total and we equally divide S bytes into N files where N is perfectly dividable by cc. Thus there are $n = N/cc$ files per concurrent transfer (i.e., per GridFTP process), and each file of size $x = \frac{S}{N}$. The transfer time T to number of files N model will be:

$$T(N) = \frac{S}{cc}a_{\text{trs}} + \frac{N}{cc}b_{\text{trs}} + \frac{S}{N}a_{\text{ck}} + b_{\text{ck}} + b_{\text{trs}}. \tag{6}$$

We use experimental data to estimate the four parameters in Equation 6, a_{trs}, b_{trs}, a_{ck}, and b_{ck}, in our environment and for different concurrency values, cc. As we previously determined (see Figure 4) that parallelism makes little difference in our low RTT environment, we fixed parallelism at four in these experiments. For each concurrency value, we fixed S=1.2 TB and used four measured (N, T) points to fit the four parameters, and then used the resulting model to predict performance for other values of N. Figure 9 shows our results. Here, the lines are model predictions, stars are measured values used to fit the model, and other dots are other measured values not used to fit the model.

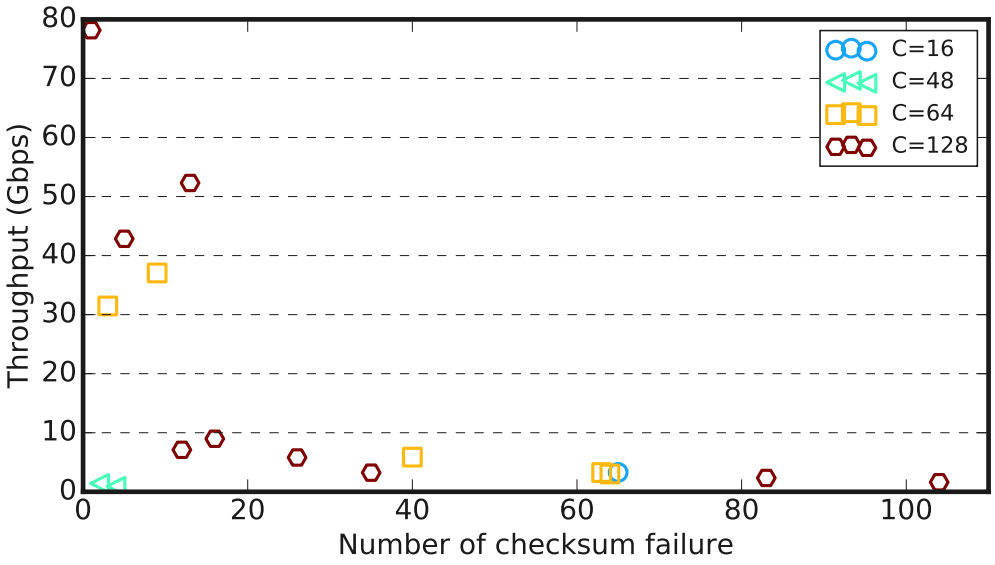

Figure 7: Data transfer performance versus number of checksum failure.

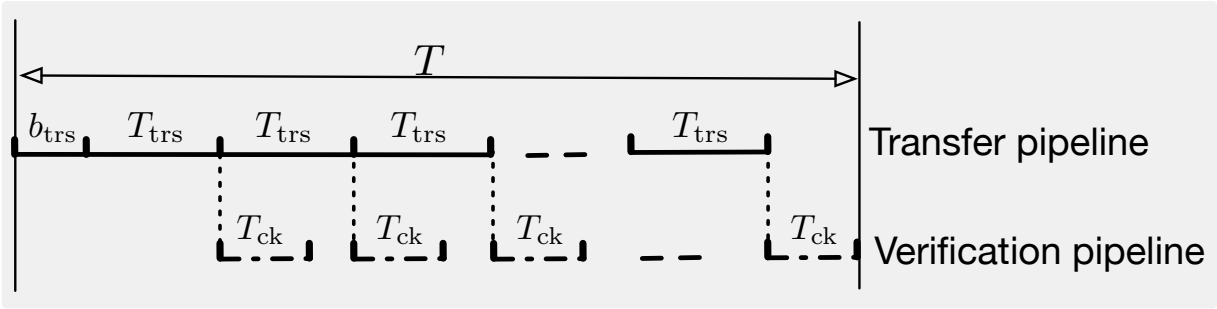

Figure 8: An illustration of file transfer and verification overlap within a single GridFTP process, as modeled by Equation 5. A sequence of file transfer and associated file verification operations are managed as independent pipelines, connected only in that a file can only be verified after it has been transferred, and verification failure causes a file retransmit.

We see that our model does a good job of predicting throughput as a function of N and cc. Thus, we can use it to determine the best file size to use when splitting a simulation snapshot.

Figure 9 shows the accuracy of the performance model, it is clear that the model can make accuracy prediction. Since the four model parameters are independent of source, network, and destination, at least four experiments are needed to fit the model, after which the model can be used to determine the best file size to split a simulation snapshot. Based on this approach, we conclude that the optimal file size is around 500 MB and it can achieve a throughput of 89.7 Gb/s with integrity verification. This throughput represents an increase of 25% compared to that obtained with the ad hoc approach, when we used a file size of 4 GB.

6. Related work

Elephant flows such as those considered here have been known to account for over 90% of the bytes transferred on typical networks [11], making their optimization important.

At the 2009 Supercomputing conference, a multi-disciplinary team of researchers from DOE national laboratories and universities demonstrated the seamless, reliable, and rapid transfer of 10 TB of Earth System Grid data [12] from three sources—the Argonne Leadership Computing Facility, Lawrence Livermore

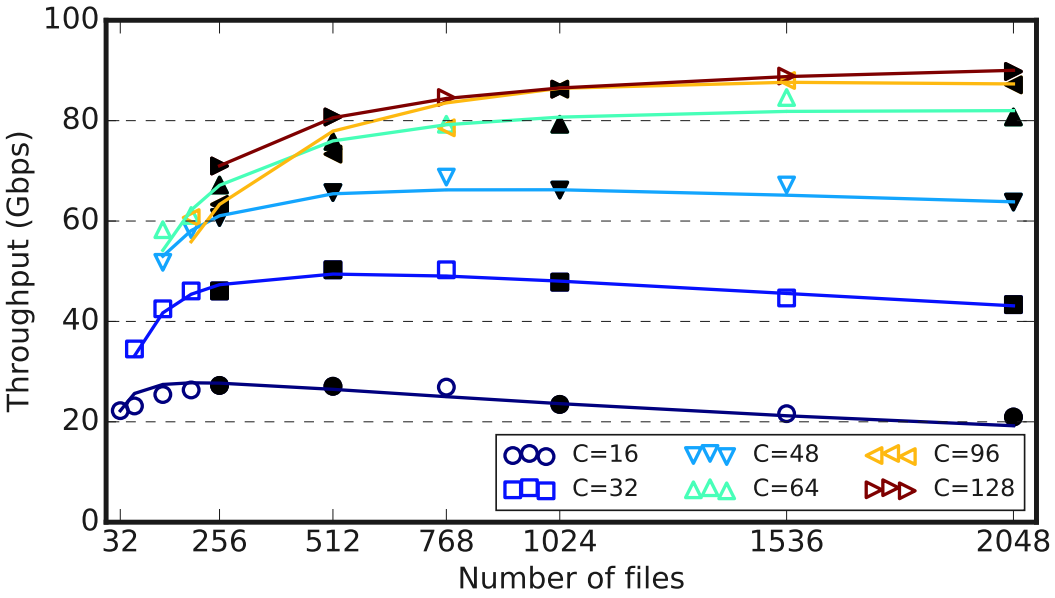

Figure 9: Evaluation of our transfer performance model when transferring a single 1.2 TB snapshot. Solid markers are points that were used to fit the model parameters shown in Equation 6.

National Laboratory, and National Energy Research Scientific Computing Center—Portland, Oregon, over 20 Gb/s circuits provided by DOE's ESnet. The team achieved a sustained data rate of 15 Gb/s on a 20 Gb/s network. More importantly, their work provided critical feedback on how to deploy, tune, and monitor the middleware used to replicate petascale climate datasets [13]. Their work made it clear why supercomputer centers need to install dedicated hosts, referred to as data transfer nodes, for wide area transfers [7].

In another SC experiment, this time in 2011, Balman et al. [14] streamed cosmology simulation data over a 100 Gb/s network to a remote visualization system, obtaining an average performance of 85 Gb/s. However, data were communicated with a lossy UDP-based protocol.

Many researchers have studied the impact of parameters such as concurrency and parallelism on data transfer performance [4, 5, 6] and proposed and evaluated alternative transfer protocols [15, 16, 17] and implementations [18]. Jung et al. [19] proposed a serverless data movement architecture that bypasses data transfer nodes, the filesystem stack, and the host system stack and directly moves data from one disk array controller to another, in order to obtain the highest end-to-end data transfer performance. Newman et al. [20] summarize the next-generation exascale network integrated architecture project that is designed to accomplish new levels of network and computing capabilities in support of global science collaborations through the development of a new class of intelligent, agile networked systems.

Rao et al. [21] studied the performance of TCP variants and their parameters for high-performance transfers over dedicated connections by collecting systematic measurements using physical and emulated dedicated connections. These experiments revealed important properties such as concave regions and relationships between dynamics and throughput profiles. Their analyses enable the selection of a high throughput transport method and corresponding parameters for a given connection based on round trip time. Liu et al. [22] similarly studied UDT [23].

7. Conclusion

We have presented our experiences in our attempts to transfer one petabyte of science data within one day. We first described the exploration that we performed to identify parameter values that yield maximum performance for Globus transfers. We then discussed our experiences in transferring data while the data are produced by the simulation, both with and without end-to-end integrity verification. We achieved 99.8% of

our one petabyte-per-day goal without integrity verification and 78% with integrity verification. Finally, we used a model-based approach to identify the optimal file size for transfers, with results that suggest that we could achieve 97% of our goal *with* integrity verification by choosing the appropriate file size. We believe that our work serves as a useful lesson in the time-constrained transfer of large datasets.

References

[1] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Lukić, S. Sehrish, W. Liao, HACC: Simulating sky surveys on state-of-the-art supercomputing architectures, New Astronomy 42 (2016) 49–65.

[2] K. Chard, S. Tuecke, I. Foster, Efficient and secure transfer, synchronization, and sharing of big data, IEEE Cloud Computing 1 (3) (2014) 46–55.

[3] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, I. Foster, Swift: A language for distributed parallel scripting, Parallel Computing 37 (9) (2011) 633–652.

[4] T. J. Hacker, B. D. Athey, B. Noble, The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network, in: Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM, IEEE, 2001, pp. 10–pp.

[5] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, The Globus striped GridFTP framework and server, in: ACM/IEEE Conference on Supercomputing, IEEE Computer Society, 2005, p. 54.

[6] E. Yildirim, E. Arslan, J. Kim, T. Kosar, Application-level optimization of big data transfers through pipelining, parallelism and concurrency, IEEE Transactions on Cloud Computing 4 (1) (2016) 63–75.

[7] E. Dart, L. Rotman, B. Tierney, M. Hester, J. Zurawski, The Science DMZ: A network design pattern for data-intensive science, Scientific Programming 22 (2) (2014) 173–185.

[8] V. Paxson, End-to-end internet packet dynamics, IEEE/ACM Transactions on Networking 7 (3) (1999) 277–292.

[9] J. Stone, C. Partridge, When the CRC and TCP checksum disagree, in: ACM SIGCOMM Computer Communication Review, Vol. 30, ACM, 2000, pp. 309–319.

[10] L. N. Bairavasundaram, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, G. R. Goodson, B. Schroeder, An analysis of data corruption in the storage stack, ACM Transactions on Storage 4 (3) (2008) 8.

[11] K. Lan, J. Heidemann, A measurement study of correlations of internet flow characteristics, Computer Networks 50 (1) (2006) 46–62.

[12] D. N. Williams, R. Drach, R. Ananthakrishnan, I. T. Foster, D. Fraser, F. Siebenlist, D. E. Bernholdt, M. Chen, J. Schwidder, S. Bharathi, A. L. Chervenak, R. Schuler, M. Su, D. Brown, L. Cinquini, P. Fox, J. Garcia, D. E. Middleton, W. G. Strand, N. Wilhelmi, S. Hankin, R. Schweitzer, P. Jones, A. Shoshani, A. Sim, The Earth System Grid: Enabling access to multimodel climate simulation data, Bulletin of the American Meteorological Society 90 (2) (2009) 195–205. doi:10.1175/2008BAMS2459.1.

[13] R. Kettimuthu, A. Sim, D. Gunter, B. Allcock, P.-T. Bremer, J. Bresnahan, A. Cherry, L. Childers, E. Dart, I. Foster, K. Harms, J. Hick, J. Lee, M. Link, J. Long, K. Miller, V. Natarajan, V. Pascucci, K. Raffenetti, D. Ressman, D. Williams, L. Wilson, L. Winkler, Lessons learned from moving Earth System Grid data sets over a 20 Gbps wide-area network, in: 19th ACM International Symposium on High Performance Distributed Computing, HPDC '10, ACM, New York, NY, USA, 2010, pp. 316–319. doi:10.1145/1851476.1851519.

[14] M. Balman, E. Pouyoul, Y. Yao, E. Bethel, B. Loring, M. Prabhat, J. Shalf, A. Sim, B. L. Tierney, Experiences with 100Gbps network applications, in: 5th International Workshop on Data-Intensive Distributed Computing, ACM, 2012, pp. 33–42.

[15] E. Kissel, M. Swany, B. Tierney, E. Pouyoul, Efficient wide area data transfer protocols for 100 Gbps networks and beyond, in: 3rd International Workshop on Network-Aware Data Management, ACM, 2013, p. 3.

[16] D. X. Wei, C. Jin, S. H. Low, S. Hegde, FAST TCP: Motivation, architecture, algorithms, performance, IEEE/ACM Transactions on Networking 14 (6) (2006) 1246–1259.

[17] L. Zhang, W. Wu, P. DeMar, E. Pouyoul, mdtmFTP and its evaluation on ESNET SDN testbed, Future Generation Computer Systems.

[18] H. Bullot, R. Les Cottrell, R. Hughes-Jones, Evaluation of advanced TCP stacks on fast long-distance production networks, Journal of Grid Computing 1 (4) (2003) 345–359.

[19] E. S. Jung, R. Kettimuthu, High-performance serverless data transfer over wide-area networks, in: 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, 2015, pp. 557–564. doi:10.1109/IPDPSW.2015.69.

[20] H. Newman, M. Spiropulu, J. Balcas, D. Kcira, I. Legrand, A. Mughal, J. Vlimant, R. Voicu, Next-generation exascale network integrated architecture for global science, Journal of Optical Communications and Networking 9 (2) (2017) A162–A169.

[21] N. S. Rao, Q. Liu, S. Sen, D. Towlsey, G. Vardoyan, R. Kettimuthu, I. Foster, TCP throughput profiles using measurements over dedicated connections, in: 26th International Symposium on High-Performance Parallel and Distributed Computing, ACM, 2017, pp. 193–204.

[22] Q. Liu, N. S. Rao, C. Q. Wu, D. Yun, R. Kettimuthu, I. T. Foster, Measurement-based performance profiles and dynamics of UDT over dedicated connections, in: 24th International Conference on Network Protocols, IEEE, 2016, pp. 1–10.

[23] Y. Gu, R. L. Grossman, UDT: UDP-based data transfer for high-speed wide area networks, Computer Networks 51 (7) (2007) 1777–1799.

Acknowledgment

This work was supported in part by the US Department of Energy under contract number DEAC02-06CH11357 and in part by the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois.